Analyses lexicale et syntaxique

L'objectif de ce TP est de vous familiariser avec les outils usuels pour l'analyse lexicale et l'analyse syntaxique. On manipulera la version Caml de ces outils, `ocamllex` et `ocamlyacc` — l'approche est sensiblement la même en C ou en Java.

On se propose de faire les analyses lexicale et syntaxique des programmes IMP.

Téléchargez l'archive `tp13.tgz` depuis la page des TPs du cours, et décompressez-la (`tar zxf tp13.tgz`).

1 S'esbaudir

Commencez par vous émerveiller du fonctionnement du programme qui vous est fourni: tapez successivement

`cd frontend`

puis

`make`

puis

`./Imp < tests/pg0.imp`

ce qui signifie que l'on lance `Imp` (notre "compilateur") sur le programme `pg0.imp` qui se trouve dans le répertoire `tests/` (pensez à utiliser la touche tabulation pour ne pas avoir à tout taper).

Faites ensuite

`cat tests/pg0.imp`

pour observer le programme qui a été donné en entrée.

2 Lire

Ouvrez dans un éditeur de texte et lisez successivement les fichiers suivants (ils contiennent des commentaires destinés à vous aider à la lecture):

- `term.ml`, qui définit les types servant à la représentation des programmes IMP.
- `lexer.mll` (*attention!* `lexer.mll`, pas `lexer.ml`), qui contient les éléments pour l'analyse lexicale.
- `parser.mly` (*attention!* `.mly`), qui contient les éléments pour l'analyse syntaxique.
- Jetez un coup d'œil rapide à `main.ml`, mais ce qu'il contient n'est pas très instructif pour nous (en particulier, c'est un fichier que vous n'aurez pas a priori à modifier).

Comprenez.

3 Agir

Le compilateur que vous avez ne fonctionne que sur `pg0.imp`. Modifiez-le afin qu'il puisse *successivement* "manger" les programmes contenus dans `pg1.imp`, `pg2.imp`, ...

Pour cela, il faudra modifier les fichiers `lexer.mll` et `parser.mly`, recompiler à chaque fois, jusqu'à ce que ça marche.

Outre du chargé de TP, vous pouvez vous aider des indications suivantes:

- – pg1: vérifiez le parenthésage de ce qu'a reconnu le programme;
  – pg2: il y a a priori plusieurs manières d'ajouter `>`;
  – pg3: vérifiez le traitement de la dernière ligne;
  – pg4: posez vous la question de `if b then if b' then c1 else c2 else c3`
- L'extension de la grammaire se fait en ajoutant des règles dans le fichier `parser.mly`. Cela peut engendrer des conflits, car dans certaines situations l'automate vu en cours (cf. le fichier `animation-yacc.pdf`) ne sait s'il doit faire *shift* ou *reduce*.
- De fait, l'erreur la plus fréquente lorsque l'on met au point une grammaire avec `ocamlyacc` est un *'shift-reduce conflict'*. Cela correspond à la présence de situations ambiguës, où plusieurs règles peuvent être appliquées pour analyser la même séquence de lexèmes.

  Lorsque ce message apparaît à la compilation, vous pouvez avoir des indications en allant regarder le fichier `_build/parser.output`, qui contient la description de l'automate qui est engendré.

Question: Profitez de votre premier conflit shift/reduce pour le comprendre, et pondre un exemple qui vous dira si l'attitude par défaut de `ocamlyacc` en pareille situation est de faire shift ou reduce.

Mais il ne faut pas laisser de tels conflits. Pour cela, on s'appuie sur la partie "priorités" du fichier `parser.mly`.

Question: Quelle situation de conflit shift/reduce est résolue par les déclarations d'associativités (`%left` et `%right` dans `parser.mly`)? À quelle action de l'automate correspondent les associativités gauche et droite?

Question: Dans le fichier d'exemple `pg3.imp`, enlevez les espaces autour du moins. Retestez ce nouvel exemple, vous devez observer un problème. À vous de modifier le lexer et le parser pour y remédier (indice: enlever le moins unaire devant les entiers dans le lexer et rajoutez le dans la grammaire).

Le serpent. Vous pouvez vérifier la robustesse de votre compilateur en lui donnant à manger ce qu'il reconnaît. Pour cela, tapez:
`./Imp < tests/pg3.imp | grep -v programme | ./Imp`
sans comprendre *(ou alors en comprenant que l'on redirige la sortie de la commande* `./Imp < tests/pg3.imp` *vers la commande* `grep`*, qui élimine la ligne de "discours", puis vers la commande* `./Imp`*).*

Question: Le fichier `term.ml` contient les fonctions d'affichage des termes. Modifier ces fonctions pour qu'elles n'utilisent qu'un nombre minimal de parenthèses. Attention à ne pas en enlever trop non plus.